Visual Information Systems - A Database Perspective

Jens Thamm, Sven Thelemann, and Lutz Wegner*
Universität Kassel, FB Mathematik/Informatik, D-34109 Kassel, Germany
email: {injt, thele, wegner}@db.informatik.uni-kassel.de

Abstract. Database theory is a mature science with many well researched and implemented techniques. Examples are the layered architecture with external, conceptual, and internal model, integrity constraints, descriptive query languages, transactions, etc. Visual Information systems pose a new challenge to the database community and database researchers must open their world for the new media and the needs of world-wide networking. At the same time projects in the area of visual information systems, spatial data, multimedia, digital libraries, etc. should look at database technology for trusted solutions. This paper exemplifies the above theme with experiences gained from the ESCHER database prototype which features visual interaction for browsing and editing all kinds of data.

Keywords: visualization, information systems, non-standard database systems

1 Introduction

Visual information systems - very much like multimedia systems - are characterized by the integrated, computer-controlled generation, manipulation, presentation, storage, and communication of independent discrete and continuous media.[14] Within this definition, discrete media include text and pixel/vector graphics while continuous media have a temporal dimension and cover digital audio and video, animation, formatted music encodings and continuously sampled scientific data.

In particular, visual information systems (VIS) are concerned with browsing and querying of information, both from the interaction (operational) side and from the presentation point of view. In this respect they are closely related to database management systems, whose purpose could be defined as keeping data non-redundantly in a persistent store, providing each user with a logical view of these data and permitting users and applications to query and manipulate the data while maintaining their global integrity.

In this paper we shall argue that both worlds are not well integrated, i.e. first generation multimedia and visual information systems often ignore trusted database concepts, like integrity constraints - as evidenced by dangling links in the Web - and, conversely, many commercial database management systems (DBMSs) offer no well-typed integration of non-textual data, relegating them to unspecified binary-large-objects (blobs) with poor interfaces.

This inadequate state of the affair is even more annoying as high resolution display devices become more affordable and high-bandwidth interconnectivity becomes more wide-spread. These technological advances will soon permit bi-directional visual information transfer anytime anywhere - termed ubiquitous computing - much in the way cellular phones are used today. Thus there is a definite technological push, yet there is also an application pull1, because visual information exchange is very effective by using the input channel with the highest bandwidth of all sensory organs2.

Unfortunately, there is an asymmetrical aspect about visual information systems: it has been argued by e.g. H. Maurer [25, 26] that humans lack an organ which is the inverse to the eye in the sense that the mouth is the inverse organ to the ear. Indeed, gestures and facial expressions seem to be our only means of producing direct visual information. Thus, we have to rely on a computer-mediated interface to “speak” visually [6]. Producing, storing and transmitting semantically rich visual information - other than based on written word - is thus considerably more difficult than producing, storing and transmitting audio data, say voice.

The remainder of this paper is organized as follows. In Sections 2 and 3 we argue in favor of a layered approach to VIS. We also will try to convince the reader that the actual data model for the conceptual schema is fairly irrelevant - some type of graph model will suffice - but that the nested relational model, also called Non-First Normal-Form (NF2) data model [1, 8, 17, 32], is the one of choice at least on the external and presentation layer.

We defend our argument in Section 4 with examples from ESCHER, our prototype database system supporting visualization in non-standard applications in engineering, science, tourism, and the entertainment industry [22, 29, 34, 37, 39, 40, 41]. ESCHER originated in 1987 as part of a joint research project with IBM Scientific Center Heidelberg and became operational in 1989 as a database editor. ESCHER is available as public domain software for

1. Creating a predicted $3.79 billion market for visual development tools by 1999 according to [33].
2. about 40 GB/day for seeing versus 400 MB/day for hearing versus 10 MB/day for reading [10]
IBM RS/6000 and HP 9000 series under OSF/Motif from our ftp site in Kassel.

Section 5 discusses our approach to object traversal, both on the presentation and on the physical level, by means of so-called fingers, which are a generalization of the well-known cursor concept. In Section 6 we will then report on our experiences with multimedia and network aspects. Section 7 summarizes the paper.

2 A Layered Architecture for VIS

One of the reasons for the success of database technology is the general acceptance of a layered architecture along the lines of the ANSI-SPARC model [36]. In this model, external (application, user) views are part of the external schema and are separated from the global logical view of the data, called the conceptual schema. This in turn is independent of the physical appearance of data, called the internal schema, which in these days also includes distribution (fragmentation, network) aspects.

Relational database systems excel at this separation with user definable views based on stored SQL-queries for the external schemas and an internal schema which hides details of the storage and access structures, operating system, implementation language and hardware. On the other hand, their ability to accommodate non-textual values is limited, which makes them second choice for multimedia developments, as is their suitability for non-standard applications involving complex objects in engineering, science, publishing and similar fields [11, 15, 27].

Object-oriented databases have more to offer in this respect (see e.g. [21] regarding suitability for multimedia). Since most of them are implemented as persistent extensions of C++, which acts simultaneously as implementation, data definition and data manipulation language, the separation of layers becomes fuzzier.

It is obvious that visual information systems should strive for this layering too. In fact, the Human-Computer-Interface (HCI) research community has always used a multiple-layer approach separating e.g. the global design model from the user model. However, there is a mismatch of terms because these separations are mostly concerned with system perception by humans. In HCI the design model is also called the logical model, which corresponds well to the logical/conceptual model in the DB-world, and is usually based on some metaphor, as shown in Table 1 below.

However, what is called the conceptual model in HCI relates to the familiar knowledge of people, i.e. their perception (understanding, view) of a system and would be termed an external model in the DB-world. The implementation of the logical design results in a physical design which is often divided into operational aspects and representational aspects [31, p. 435]. Again there is a slight mis-

---

### Table 1: Examples of applications and associated metaphors [31, p. 149]

<table>
<thead>
<tr>
<th>Application</th>
<th>Metaphor</th>
<th>Familiar knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating environment</td>
<td>The desktop</td>
<td>Office tasks, file management</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>Ledger sheet</td>
<td>Columnar tables</td>
</tr>
<tr>
<td>OO environments</td>
<td>Physical world</td>
<td>Real-world behaviour</td>
</tr>
<tr>
<td>Hypertext</td>
<td>Notecards</td>
<td>Flexible organization of structured text</td>
</tr>
<tr>
<td>Learning environments</td>
<td>Travel</td>
<td>Tours, guides, navigation</td>
</tr>
<tr>
<td>File storage</td>
<td>Piles</td>
<td>Categorizing objects in terms of urgency, projects, ...</td>
</tr>
<tr>
<td>Multimedia environments</td>
<td>Rooms assoc. with media</td>
<td>Spatial structure of buildings</td>
</tr>
<tr>
<td>CSCW</td>
<td>Multi-agents</td>
<td>Travel agents, butlers and other serving roles</td>
</tr>
</tbody>
</table>

---

### Table 2: A four layer architecture for Visual Information Systems

<table>
<thead>
<tr>
<th>Layer</th>
<th>Purpose of Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation/dialog model</td>
<td>Result representation, interaction setting</td>
</tr>
<tr>
<td>External model</td>
<td>User selections and projections, the content filter</td>
</tr>
<tr>
<td>Logical model</td>
<td>Global concept and its set of permissible operations</td>
</tr>
<tr>
<td>Internal/physical model</td>
<td>Implementation aspects</td>
</tr>
</tbody>
</table>

At the presentation layer, form should follow content, i.e. different data need different display styles. Examples of different styles are graphs, charts, tables and forms.
Desired quality of service comes into play for continuous media, quantity of the data is important as some styles don’t scale well for thousands of data tuples. In general, visual information systems should offer a variety of representations but with a core of uniform interaction principles. In the following we argue that a generalized cursor, called a *finger* in ESCHER, constitutes such a trans-presentation paradigm.

The **external schema** below the presentation layer provides the user with a restricted view of the world. This filter concerns both attributes which are omitted, corresponding to projections in the relational world, as well as instances which are omitted based on certain values, corresponding to selection in a relational DBMS.

The particular view of the world is dependent on the role of the user or his/her fields of interest, e.g. in a hospital world, a surgeon gets x-ray data but without billing information and restricted to those patients who will undergo surgery the next day, while the accounting clerk needs no x-ray pictures and might have access to only those patients who have just left hospital. For visual information systems we suggest a QBE-like query facility like ESCHER’s query subsystem.

For the **logical schema**, designers of visual information systems must decide on one model which has enough expressive power to handle all the desired interconnections and operations, yet is concise enough to be described in algebraic terms. Finally, the model must yield an efficient implementation. We shall argue that extended relational systems form a useful base but further research is needed, in particular to encompass the temporal aspects (streams) of continuous media.

Finally, there is again a wide choice of implementation techniques for the **physical layer**. New technologies, like scripting languages for interface design and networking, offer new possibilities. At the very core, they must be able to traverse and update the trees and graphs which form the complex objects which in turn embody the stored information of visual information systems. Below we will show that the cursors introduced on the presentation layer translate nicely into stacks of node identifiers and that these stacks can be manipulated efficiently with a small set of operations implementable in a variety of software systems.

To close this Section, let us consider a first generation hypertext systems like the World Wide Web (W3) [5] and see how it fares within this schema.

Clearly the logical model is the hypertext model, based on the notecard metaphor, augmented by graphical elements which can be tied into the documents. The hypertext model in turn can be considered as a simple directed graph (permitting cycles which causes some problems for internet crawlers). The permissible operations are simple: when placed at a node (a document), all directly reachable nodes are represented by links which are „click-able“ words or maps inside the document. Links in W3 are thus directly anchored inside the document and contain the destination node address.

From a database point of view, there is too little separation from the physical model, because the links are practically hard-wired into the documents, can be directly manipulated by the owner of the document, are visible to anyone and, most important of all, are uni-directional, giving frequently rise to integrity violations - the infamous „404 Not Found. The requested URL /... was not found on this server.“

On the other hand, there are clear attempts to separate the logical from the physical model: users are encouraged to write their HTML documents in a logical style, e.g. to use `<em>`, `<strong>`,¹ ... instead of `<b>`, `<i>`, `<a>`, `<tt>`² ... Similarly, there is a move to support those users who cannot or do not want to load pages which are overly rich decorated with large pixel components. Moreover, the physical layer presents no problems at the moment, because HTML has such simple syntax and semantics that porting browsers to all kinds of hardware and operating systems seems to be sufficiently easy. Even for dynamic, interactive extensions like Sun’s Java [35], various ports have been announced. We expect this to change when more and more features are incorporated into those languages and incompatibility increases.

Moving up from the logical model to a user model, we note that W3 has little support for selection or projection of domains of interest, i.e. everybody navigates (logically and physically) in the same global graph, which leads to the well-known „lost in dataspace“ phenomenon. Information filters, search engines and location servers were added later.

Finally, on the presentation layer, we have browsers like NetScape and Mosaic and a number of editors for different machines, all with almost identical capabilities and very similar „Look&Feel“.

Second-generation hypermedia systems, like Hyper-G [25], try to improve on this by separating links from documents, making links bi-directional, permitting other document display styles (e.g. PostScript), offering clusters and collections, incorporating search facilities and much more.

### 3 Which Logical Model?

If visual information systems should be based on a layered model similar to the four layer architecture from above, then the next question is: „What logical data model is best suited to visual information systems?“

We think that there are basically three choices:

- a set and value based model:

1. emphasis and strong emphasis
2. bold, italics, underline, typewriter font
• a graph and object based model;
• a hybrid of both

The first model could be supported by a flat or nested relational DBMS while the second would go into the direction of a network and object-oriented DBMS.

To illustrate the point, consider the flight plan of an airline carrier, restricted to seven major Australian cities and direct connections only. In an object-oriented DBMS this might be implemented using two classes, Flights and Cities, both inherited from some suitable super-classes.

CLASS Cities SET OF City
TYPE City TUPLE OF(
  Code: STRING
  Name: STRING
  Timezone: REAL)
CLASS Flights SET OF Flight
TYPE Flight TUPLE OF(
  From: City
  To: City
  Days: STRING
  departs: TIME
  arrives: TIME
  FlightNo: INTEGER
  Restrict: STRING)

A particular instance of a flight with object identifiers OID\(_i\) and OID\(_j\) relating a particular flight to its origin and destination cities is shown in Figure 1.

![Fig. 1: Instances of flights in an OO-model](image)

In a relational model, flights and cities would form two separate tables with the airport code acting as foreign key in the flights table (Tables 3 and 4).

Avoiding the religious war between these two factions, we claim that both models are attractive, as long as they support
• navigation
• declarative queries
• integrity constraints
• type inheritance
• recursion

Indeed, some object-oriented DBMS support SQL-like queries and some forms of integrity control. Other graph-based models support query and update languages like HNQL [24].

On the other hand, RDBMS can support cursors to allow navigational modes and have been extended for recursion. Also, SQL-3 [13] will feature a type hierarchy with inheritance and methods (stored procedures).

At the presentation layer, both graphs and flat tables have no attractive visualization. Graphs composed of nodes and arcs are excellent for a limited number of instances (about a dozen) and for showing relations between entity types or type hierarchies in general. In our example we can show all (seven) cities which have direct connections in a graph mapped onto the outline of the Australian continent (Figure 2).

However, the graph is a condensation of 260 direct flights between these seven Australian cities taken from the Ansett Australia Travel Planner of April 1995. There is absolutely no way that the graph could convey the information from this 260 row table, not to mention the 2000+ connections in the original Ansett Travel Planner.

Similarly, flat tables of this size are not suitable for browsing either but relational systems provide query facilities for precisely this case, possibly with some form based result display.

<table>
<thead>
<tr>
<th>Table 3: Flights</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRO</td>
</tr>
<tr>
<td>ADL</td>
</tr>
<tr>
<td>DRW</td>
</tr>
<tr>
<td>DRW</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4: Cities</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
</tr>
<tr>
<td>ADL</td>
</tr>
<tr>
<td>BNE</td>
</tr>
<tr>
<td>CNS</td>
</tr>
<tr>
<td>DRW</td>
</tr>
<tr>
<td>MEL</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

...
4 A Tabular Approach to VIS

4.1 Introducing ESCHER

Keeping with the flights example, consider the nested table with a window over the actual full table of 260 direct connections shown as a screen-shot taken from our database editor ESCHER (Figure 3). Note that we have chosen to also include destinations not served by direct flights in which case the set of flights is empty (denoted by {}).

The screen shows the normal layout of visualization with a schema displayed above a table. The schema defines a set of connections, i.e. CONNECTIONS is a set-valued attribute taking tuples with 4 attributes as members: a city CODE, the city NAME and its time zone TZONE (GMT+). The fourth attribute is again set-valued containing all destinations with destination code, name and flight time.

In ESCHER, a schema may define several tables in the same sense in which several papers may be written with the same layout sheet. Schemas and tables are grouped into applications. Relations between schemas, tables and applications are controlled by tables inside the special application system.

Returning to Figure 3, we see that the display of the nested relational table follows certain style rules: we draw separating lines between tuples within a collection if the tuples contain at least one non-atomic (not all text, integer, real, Boolean, ...) attribute and we do not draw vertical lines through empty sets or through null-valued sets or lists (ordered collections) which are indicated by \{ \} and <???>", respectively. Note also that an empty set of flights has a null-valued flight time indicated by „?s?“.

4.2 Navigation

Visual information systems must support browsing and navigation. As a visual database editor, ESCHER provides access to data by means of so called fingers. They generalize the cursor paradigm in graphical and text editors. On the graphical display, a finger is reflected by a colored area which corresponds to the object a finger is currently pointing at. In a table more than one finger may point to objects, one of which is the active finger and is used for navigating through the table.

In Figure 3, finger F1 is the active one. The colored areas indicate that F1 is positioned on a DESTINATION-tuple. Note that there are actually two colored areas: apart from the data object, the corresponding structure in the schema part is also shaded. The mode is Browse because atomic values cannot be entered for complex attributes. When the user descends to the atomic fields of this complex attribute, the mode-indicator will change to Edit-mode and the data entry widget at the lower left will become activated.

Essential operations on fingers are the navigational operations "going into an object" (In, i.e. descending into the next deeper nesting level), "out to the surrounding object" (Out), "to the next object" (Next, staying on the same nesting level), and "to the previous object" (Prev, Back). At the interface, the user navigates through the instances using these basic finger operations, either by clicking on the buttons in the left corner of the window, by use of the ESC-, ENTER- and arrow-keys, or by directly clicking with the mouse into the table. In the latter case, the finger „jumps“ to the atomic field in which the mouse cursor is positioned. If the user wants to point to the surrounding (complex) object, the user needs to hold the left button pressed and drag the mouse. The finger will immediately move Out to the smallest surrounding object which contains the start and end point of the mouse movement.

Internally, a mouse click gets translated in a sequence of finger operations operating on schema and table trees as explained in Section 5. The technically difficult aspect of this direct interaction interface is „catching“ the mouse cursor and quickly redrawing the visible portion. The reason for the difficulty is that (1) the tables contain variably sized objects with coordinates changing due to insertions and deletions, and that (2) the tables can have millions of

---

1. The null value semantics of relational database theory extends naturally to nested relational databases; an empty set of flights indicates certainty about the non-existance of such flights while a set-valued null indicates that the proper setting is not known or inapplicable. A null-valued tuple for the set of flights in turn would indicate that a flight exists but its data are not known.
tuples and are thus never stored, not even temporarily, as
bitmaps. Details of GUI-algorithms developed for ESCHER
must be skipped here for lack of space.

Atomic attributes are updated by going once more
"into" an atomic object (mouse double click) and
modifying the current value.

Atomic and complex objects can be edited using
Insert, Delete, Backspace, Cut-and-Paste with a clipboard, etc.
Note that operations involving the clipboard work with a
sub-schema which is established when the first object is
moved into it and that objects which are appended to the
clipboard must have matching types.

4.3 Queries
Visual information systems must support some kind of
querying mechanism (see [3] for a survey). Ideally, this
facility is integral part of the system and not set on top as
the search engines for W3.

For nested relational systems there are SQL-extensi-
ons, like HDBL [8]. Other commercial systems, like
SAP’s ABAP, start to support complex objects and SQL-3
extends the relational model in a similar fashion. However,
the language-based approaches are not suited to the novice
user and therefore ESCHER has implemented a query faci-
lity modeled after QBE (Query-By-Example [42, 28]).

As an example, consider a query which asks for the
flight time of direct flights from Sydney to Brisbane or
Darwin based upon a smaller table (Figure 4).

\[
\{ \text{y.FTIME} \mid \text{x in DFLIGHTS}, \text{y in x.TO:} \\
\text{x.FROMCITY = "Sydney" and (y.TO = } \text{"Brisbane" or y.TO = "Darwin")}\}
\]

Formulating and running the query in ESCHER involves
the following steps (for details see [40]):
(i) A (nested) view table is created or a given nested
base table is taken.
(ii) The table is displayed and the user starts a query by
clicking on the BeginQ button. This step corresponds to the
FROM-part of a SELECT-FROM-WHERE query.
(iii) A dependent window appears with a QUERY template and schema. The user edits the template to specify the selection predicates in a QBE-like fashion. Step (iii) corresponds to the WHERE-part of a SELECT-FROM-WHERE query.

(iv) For the SELECT-part of the query, the user has the choice between two modes of query result display:

- **Search mode**: The user positions another finger to the substructure of interest in the QUERY schema. When the query is executed, the active finger in the window for the query table becomes the *query finger*. It is positioned on the first data object that “matches” with the select conditions as specified in the QUERY template. Its structure is constrained by the structure given by the active finger in the QUERY schema. By clicking on the Next button the user can move the query finger to the next found data object and so on until the end of the table is reached. This is much in the tradition of well-known search utilities for text editors.

- **Filter mode**: The result of the query is shown in a RESULT table displayed in a separate window and only the “matching” tuples or values are filtered out. This is the way most database systems present the result of a query to the user.

(v) The user clicks on the *EndDef* button in the QUERY window. The query is then executed by clicking on the *RunQ* button.

The *search mode* is particularly suitable for a browser like ESCHER because it shows the information in context which is typical for visual information systems where precise queries and answers are often not very useful; consider e.g. asking for flights departing between 8 and 10 am which might not return any good connections, but looking at the neighborhood reveals that there is a direct flight at 10.05 which would have otherwise been missed.

Finally, it should be mentioned that queries can be saved for future reuse and that a query can be continued with the result finger from the previous query setting the scope for the next query.

5 Finger Operations as Common Ground

ESCHER is implemented using a number of self-referential methods, e.g. a circular fix-point meta-schema which describes all schemas including itself. Details on the implementation aspects may be found in [37, 38, 39], but we hope that our choice of name for the editor has now become apparent.

It has been argued by other authors [16] that self-referential methods should also extend to the interface which should be stored inside the database. Systems built upon a persistent object-oriented programming language can integrate the interface more easily as both systems share the same inheritance hierarchy and can be brought together through compiler and linker.

In ESCHER the situation is different. ESCHER is neither implemented in an object-oriented language nor is it based on an existing DBMS. Rather it is written from scratch in ordinary C. Thus what appears as a complex object (nested table) on the presentation level is implemented as a complex object on the physical level, namely mostly as a tree with additional arcs for links and cyclic structures like in the meta-schema effectively rendering it a directed cyclic graph. Figure 5 below show the tree for the table from the small example in Figure 4. A similar tree for the schema of the table exists but is not shown here.
As can be seen from the figure, fingers correspond to stacks with node addresses pointing to a path inside the tree. The addresses are invariant and thus similar to object identifiers, but not quite so detached from physical addresses, because they are swizzled at run time to become main memory pointers. Details of this very efficient scheme can be found in [39].

Finger operations are implemented by a library of so-called Object Manager Operations. Going into an object corresponds to a push-operation (since the stack grows with going down the tree) while going out corresponds to a pop-operation.

Fingers and finger operations are used everywhere in ESCHER: to sort and purge duplicates, to draw the schema and table, to compute the depth of trees, to perform queries, or in interactive schema definitions.

Separating schema from table information and treating both as trees is certainly the cleanest way in the long run. In a recent port of ESCHER’s interaction paradigm to ABAP, SAP’s 4GL application and implementation language, this proved to be difficult because schema information was contained in flat tables and had to be collected at run-time.

Another point to contemplate is ESCHER’s present lack of a scripting language which would give rise to stored (interpretable) scripts acting as methods for their enclosing objects. This has led to considerations of replacing OSF/Motif as graphical interface by Tcl/Tk which - with suitable extensions - could also act as scripting language for finger operations.

Finally, fingers and finger operations introduce concurrency. When fingers are assigned to distinct users, tables become the workspace for computer supported cooperative work (CSCW). This requires extensions of the transaction model with its ACID (atomicity, concurrency, isolation, durability) principles.

In [20], Korth argues that the focus on workflow management created further “evolution of the transaction model [which] is a shift from machine-oriented concepts to human-oriented concepts.” He continues to point out that the key ingredient is interaction and that the human performance determines the throughput of the system, i.e. “the quantity to be maximized is human information transfers per second (HITS)” as opposed to transactions per second (TPS) [20].

ESCHER addresses this area with the concept of negotiated transactions which emphasize the fact that users are visually aware of concurrent activities within the table. Typical examples are concurrent reservations, bidding, etc. Further details may be found in [41].

### 6 Forms, Graphs and Multimedia Extensions

Visual information systems are about displaying, searching/browsing, editing data. Individual data items and even collections of data items by themselves are meaningless: a picture of a person is meaningless without knowing who the person is or why the picture is shown. A set of sale items given only by their sale number carries little information content without item description and item price.

The art of visual information display is then to present an entity with its relevant attributes together with any other entities which are related in a particular context. Examples are:

- an x-ray of a person together with textual data describing this person together with data on when and way the picture was taken;
- order number, description, and price of a sales item in a catalogue together with picture and stored procedure acting as ordering agent for the catalogue distributor;
- departure and arrival times for flight connections between two towns and a particular airline and day;
- a set of (off-line) video clips of holiday locations with availability information obtainable on-line; etc

This short list of examples also shows that - even under the restriction that the display is to be in one or several (possibly overlapping) windows on a single color CRT screen - there are many suitable display styles, e.g. graphs, tables, forms, indented lists, 3-D-walls, etc. However, it is clear that certain styles are infeasible given the amount of data which should be displayed simultaneously.

Until now we have stressed the tabular display as the dominant representation method which corresponds to our logical model and was also the first interface style implemented in ESCHER. However, it is clear that any of the
other display methods - graphs, tables and forms - are equally important. However, we believe they are geared to different information needs.

- Graphs express relations between entity types by drawing an arc between the nodes representing the classes.
- Tables express relations between entities by placing attributes of the involved entities and attributes describing the relationship next to each other in a single row. This style favors large amounts of instance data (as opposed to class data). Tables also provide a means to look at information in the “neighborhood”. They are limited in the number of columns that can be conveniently displayed and nested tables basically suffer from the same problem.
- Forms express relations between entities by showing all relevant attributes of the involved instances in a single vertical and horizontal area. Columns can now be distributed over the entire window, not constrained by horizontal juxtaposition.

Both tables and forms can provide fields for non-textual data like pixel pictures, vector graphics, sound, and video. In the case of ESCHER, our prototype features import, export, storage, and display of GIF (up to 256 colors), JPG (true color) and XBM (bitmap) pixel pictures. As an example, Figure 6 shows a nested table for a hotel information system with the cursor positioned on a hotel in Hamburg. The placement of the cursor might result from running a query formulated in a QBE-like fashion with “Hamburg” entered in the city-field as explained in Sec. 4.

Of course, a pixel-valued attribute can participate in the usual editor operations, like cut-and-paste. It could also be moved to a QBE-template to serve as search criteria except that search conditions for pictures have not been defined. Also the usual finger operations could operate inside a pixel picture, i.e. to clip part of the picture and to zoom (stepwise IN/OUT finger operation). Picture data are of

![Fig. 6: Screen shot for hotel application](image-url)
little interest for query formulation unless particular algorithms for feature extraction are considered which are outside the scope of this paper.

On the other hand, pictorial data can easily be used to enhance the information content of query results. The obvious case is to make the picture(s) integral part of the selected tuples, as with x-ray pictures in medical information systems, floor-prints and pictures of houses for sale in the real estate business, or hotel pictures in a tourist information system. Depending on the weight of the picture, the remaining textual information can be considered as an annotation of the picture which can guide the search or conversely, the picture is simply an add-on.

Continuing along this line of design, a pixel picture can be used to transform textual information, e.g. a hotel location when a street plan is added to the city tuple in the hotel example above. Moving the finger to a particular hotel creates a corresponding marker in the street plan. Again, this falls well within ESCHER’s tabular visualization paradigm but needs extra application support. Other interesting interaction and display styles for image-browsers can be found in [30].
Finally, the greatest growth in visual information systems is expected in network browsing, e.g. in the Web. As a preliminary step, we have implemented ESCHER’s display style for NF² tables using HTML (Figure 7). As for interaction and query formulation, however, HTML provides insufficient expressiveness to support ESCHER’s finger paradigm. Interested readers should visit our WWW-server at http://gretel.db.informatik.uni-kassel.de. 

Similarly, integration of vector graphics into nested tables with simultaneous movement of fingers both in the table and the graph proves to be a powerful extension as shown in [37].

On the browser side, interfaces similar to ESCHER which are suitable for the casual user are the form-based approaches [17, 18]. Other browsers navigate along the arcs of a graph which represent entities and relationships or class hierarchies [2, 7, 23]. They do not support the skeleton approach of QBE. An abstract language proposal for specifying interfaces to nested relational databases can be found in [12].

7 Conclusions

We argued that visual information systems should be based on an architecture with at least four layers:

- presentation model
- external model
- logical model
- physical model.

At the logical/conceptual level we favor a hybrid model combining the best features of set-/value-based and graph/object-based proposals. Graph-based features are needed for recursion and for navigation across different collections. Value-based features permit descriptive queries which form the basis for filtering the information into views for the external user model. Thus, nested relational algebra augmented with the concept of (typed) links could form a conceptual base for visual information systems.

At the presentation level they allow intuitive browsing and editing for novice users. Unlike graphs, nested tables scale well from a dozen to several thousand tuples. Given an appropriate nesting, users can browse over large distances when stepping on the outer level, yet can dive into details at any time.

Together with an ad-hoc query facility, NF² editors can also act like an autopilot for flying over “data space”. By entering appropriate coordinates, ESCHER will direct the user to the target area from where he or she can navigate to make “a visual landing” on a specific data tuple.

Other forms of presentation, in particular forms and hyper-documents, are equally important and are planned for future releases. Finally, integration of non-textual data and use of nested tables in the Web has also been demonstrated. 

Last, but not least, a generalized cursor concept, called fingers, forms the unifying principle from physical up to interaction level. Finger operations can be implemented in various languages for various systems, as evidenced in a recent port to a large business software system. In particular they could be implemented as extensions to scripting languages - like Tcl/Tk or Java - which will have a great influence on the shape of visual information systems of the future.

References


